CAPITOLO 22

Finestre di dialogo

di K. C. Hopson

IN QUESTO CAPITOLO

Questo capitolo si concentra sulla classe `Dialog`, che costituisce la base per creare finestre di dialogo in Java. La classe viene descritta per mezzo di diversi esempi di finestre di dialogo che forniscono suggerimenti su come utilizzare alcune delle funzionalità più oscure dell'Abstract Windowing Toolkit (AWT). La classe `FileDialog` viene discussa alla fine del capitolo.

Per comprendere a fondo la classe `Dialog`, è importante analizzare come funzionano le classi `Window` e `Frame`. Il capitolo inizia con una breve panoramica su queste due classi.

Finestre e frame

La classe `Window` viene utilizzata nell'AWT per creare finestre a comparsa che vengono visualizzate al di fuori della normale area del browser assegnata a un applet. Derivando dalla classe `Container`, la classe `Window` può contenere altri componenti. A differenza dei componenti degli applet, che sono direttamente legati a una pagina del browser, le classi `Window` non sono limitate a un'area predefinita dello schermo. È possibile modificare le dimensioni degli oggetti delle finestre in base alle specifiche necessità; l'AWT può eseguire questa regolazione automaticamente per mezzo del metodo `pack()` della classe `Window`, che funziona con il gestore di layout di `Window` (in base alle impostazioni predefinite, questo è `BorderLayout`) per ottenere la presentazione ottimale della finestra in base ai componenti contenuti e alla risoluzione dello schermo. Di norma, `pack()` viene richiamato prima che

venga visualizzata una finestra. Le finestre generalmente non sono visibili finché non viene richiamato il metodo show(), inoltre vengono rimosse dallo schermo e le relative risorse vengono rilasciate quando viene richiamato il metodo dispose().

La classe Frame estende la classe Window aggiungendo una barra del titolo, un bordo per modificare le dimensioni e il supporto per i menu. Per la maggior parte delle piattaforme delle GUI, la barra del titolo del frame è legata alle caselle di controllo del sistema, ad esempio quella per la riduzione a icona, per l'ingrandimento o per la rimozione. Di conseguenza, la classe Frame ha tutti gli elementi necessari per fare in modo che un applet assomigli a un'applicazione "reale" completa di menu e di controlli di sistema.

Il Listato 22.1 presenta un semplice applet, Frame, che cambia il titolo in base al pulsante su cui viene fatto clic. La classe dell'applet, FrameTestApplet, si limita essenzialmente ad avviare il frame principale, FrameTitles. La classe di ascolto principale, FrameTitleListener, ascolta semplicemente i comandi di azione e imposta il titolo del frame in base al nome del comando. Il costruttore stesso del frame, FrameTitles(), inizia richiamando il supercostruttore. L'unico parametro per il costruttore è la didascalia visualizzata sulla barra del titolo. Quindi viene impostato il layout per il frame; il layout predefinito è BorderLayout, ma poiché in questo caso si utilizza una matrice con una griglia di 3x2, viene utilizzato il layout GridLayout. Di seguito vengono aggiunti i pulsanti, con nomi che rappresentano il nuovo titolo del frame. Ogni pulsante è collegato alla stessa istanza di FrameTitleListener; quando si fa clic sul pulsante, viene richiamato l'ascoltatore e viene cambiato il titolo del frame. Dopo aver aggiunto tutti i componenti, al frame viene collegato un altro ascoltatore, FrameListener, che semplicemente rimuove il frame quando l'utente cerca di chiuderli. Infine, viene richiamato il metodo pack() per ottimizzare il posizionamento dei pulsanti. Poiché il posizionamento ottimale risulta essere una piccola cornice (sei pulsanti posizionati uno vicino all'altro intorno al testo dell'etichetta non occupano molto spazio), viene richiamato il metodo resize() per aumentare le dimensioni di quest'ultima. Per ultimo, il frame viene visualizzato richiamando il metodo show().

Listato 22.1 *Codice per l'applet Frame che modifica il titolo del frame.*

```
import java.awt.*;
import java.awt.event.*;
import java.lang.*;
import java.applet.*;
// Questo applet avvia il frame per mostrare
// diversi titoli...
public class FrameTestApplet extends Applet  {
   public void init() {
      // Crea il frame con un titolo...
      new FrameTitles("Titoli frame");
   }
}
// Ascolta modifiche al titolo
class FrameTitleListener implements ActionListener {
   Frame fr;  // Frame da utilizzare...
   // Costruttore...
```

```
    public FrameTitleListener(Frame fr) {
        this.fr = fr;
    }
    // Gestisce l'azione da compiere...
    public void actionPerformed(ActionEvent e) {
        // Determina l'azione eseguita...
        String selection = e.getActionCommand();
        fr.setTitle(selection);
    }
}
// Gestisce eventi di chiusura...
class FrameListener implements WindowListener {
    Frame fr;  // Frame da utilizzare...
    // Costruttore...
    public FrameListener(Frame fr) {
        this.fr = fr;
    }
    // Chiude la finestra...
    public void windowClosing(WindowEvent e) {
        fr.dispose();
    }
    public void windowOpened(WindowEvent e) {
    }
    public void windowClosed(WindowEvent e) {
    }
    public void windowIconified(WindowEvent e) {
    }
    public void windowDeiconified(WindowEvent e) {
    }
}
// Frame per consentire all'utente di scegliere
// diversi titoli da visualizzare...
class FrameTitles extends Frame {
    // Crea il frame con un titolo...
    public FrameTitles(String title) {
        // Richiama il costruttore della superclasse...
        super(title);
        // Crea un layout a griglia per i pulsanti...
        setLayout(new GridLayout(3,2));
        // Crea l'ascoltatore...
        // Aggiunge i pulsanti per scegliere i titoli...
        FrameTitleListener listener = new FrameTitleListener(this);
        Button b = new Button("Titolo frame #1");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #2");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #3");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #4");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #5");
        b.addActionListener(listener);
```

```
        add(b);
        b = new Button("Titolo frame #6");
        b.addActionListener(listener);
        add(b);
        // Un ascoltatore per chiudere il frame...
        addWindowListener(new FrameListener(this));
        // Visualizzazione...
        pack();
        resize(300,200); // Dimensione ragionevole...
        show();
    }
}
```

Introduzione alla classe Dialog

Come la classe `Frame`, anche la classe `Dialog` è una sottoclasse di `Window`, ma le finestre di dialogo differiscono dai frame per un paio di lievi differenze. La più importante è che le finestre di dialogo possono essere *modali*, vale a dire che, quando viene visualizzata una di queste finestre di dialogo, l'input in altre finestre dell'applet è bloccato finché la finestra di dialogo viene rimossa. Questa funzionalità è determinata dallo scopo generale delle finestre di dialogo, che è quello di dare all'utente un avvertimento o di permettergli di prendere una decisione prima che il programma continui. Nonostante esista il supporto per le finestre di dialogo *non modali*, la maggior parte è modale.

Esistono tre costruttori per la classe `Dialog`. Tutti e tre utilizzano come parametro iniziale un oggetto `Frame`, che indica il possessore del costruttore. Due costruttori utilizzano un ulteriore parametro `String` per il titolo e un costruttore utilizza un terzo parametro `boolean` per indicare se la finestra di dialogo è modale o meno. Molte delle finestre di dialogo create sono modali, vale a dire che accettano l'input dall'utente prima di essere distrutte.

Il Listato 22.2 mostra il codice di una variante dell'applet presentato nel Listato 22.1; in questa seconda versione, per modificare il titolo del frame viene utilizzata una finestra di dialogo. La classe di un frame, chiamata `FrameMenuTest`, e il suo ascoltatore, `FrameMenuListener`, funzionano con un menu che ha altre opzioni: mostrare la finestra di dialogo o uscire utilizzando il metodo `dispose()`. La finestra di dialogo è chiamata `ChangeTitleDialog`. Si noti che, come nell'esempio del frame, questa finestra di dialogo ha due classi di ascoltatori.

L'aspetto principale da notare nel codice di questo listato è il modo in cui viene istanziata la finestra di dialogo nella classe `FrameMenuListener` (il primo parametro è `FrameMenuTest`):

```
ChangeTitleDialog dlg;
dlg = new ChangeCursorDialog(fr,true,"Cambia il cursore");
```

Quando arriva il momento di visualizzare la finestra di dialogo, la si può dichiarare nel seguente modo:

```
dlg.show(); // Rende la finestra di dialogo visibile
```

Si noti la somiglianza della finestra di dialogo del Listato 22.2 con il frame del Listato 22.1.

Listato 22.2 *Codice per la classe Dialog che modifica il titolo del frame.*

```
import java.awt.*;
import java.lang.*;
import java.applet.*;
import java.awt.event.*;
// Questo applet avvia il frame
// che fornisce un menu per l'impostazione del titolo...
public class DialogTestApplet extends Applet  {
   public void init() {
      // Crea il frame con un titolo...
      new FrameMenuTest("Test basato su menu");
   }
}
// Ascolta cambiamenti nel titolo del frame
class FrameMenuListener implements ActionListener {
   // Finestra di dialogo per cambiare il titolo...
   ChangeTitleDialog dlg;
   Frame fr;  // Frame da utilizzare...
   // Costruttore...
   public FrameMenuListener(Frame fr) {
        this.fr = fr;
      // Istanzia la finestra di dialogo...
      dlg = new ChangeTitleDialog(fr,true,"Cambia il titolo");
   }
   // Gestisce l'azione da eseguire...
   public void actionPerformed(ActionEvent e) {
       // Determina l'azione eseguita...
       String selection = e.getActionCommand();
       if (selection.equals("Esci"))
            fr.dispose();
       // Altrimenti richiama la finestra di dialogo...
       if (selection.equals("Cambia il titolo"))
            dlg.show(); // Rende la finestra di dialogo visibile...
   }
}

// Frame che crea una finestra di dialogo che
// cambia il titolo
class FrameMenuTest extends Frame {
   // Crea il frame con un titolo...
   public FrameMenuTest(String title) {
      // Richiama il costruttore della superclasse...
      super(title);
      // Aggiunge i menu...
      // Per prima cosa crea la barra dei menu
      MenuBar mbar = new MenuBar();
      setMenuBar(mbar); // La aggancia al frame...
      // Aggiunge il sottomenu File...
      Menu m = new Menu("File");
      mbar.add(m);  // Lo aggiunge alla barra dei menu
      // Crea ascoltatore di eventi per gli elementi di menu...
      FrameMenuListener listener = new FrameMenuListener(this);
      // Aggiunge una finestra di dialogo al sottomenu...
      MenuItem item = new MenuItem("Cambia il titolo");
      item.addActionListener(listener);
      m.add(item);
```

```
        // Aggiunge un separatore
        m.addSeparator();
        // Aggiunge Esci al sottomenu...
        item = new MenuItem("Esci");
        item.addActionListener(listener);
        m.add(item);
        // Visualizzazione...
        pack();
        resize(300,200); // Dimensione ragionevole...
        show();
    }
}
// Ascolta cambiamenti nel titolo
class DialogTitleListener implements ActionListener {
    Frame fr;  // Frame da utilizzare...
    // Costruttore...
    public DialogTitleListener(Frame fr) {
         this.fr = fr;
    }
    // Gestisce l'azione da eseguire...
    public void actionPerformed(ActionEvent e) {
        // Determina l'azione eseguita...
        String selection = e.getActionCommand();
        fr.setTitle(selection);
    }
}
// Gestisce eventi di chiusura...
class DialogListener implements WindowListener {
    Dialog dlg;  // Finestra di dialogo da utilizzare...
    // Costruttore...
    public DialogListener(Dialog dlg) {
         this.dlg = dlg;
    }
    // Chiude la finestra...
    public void windowClosing(WindowEvent e) {
         dlg.dispose();
    }
    public void windowOpened(WindowEvent e) {
    }
    public void windowClosed(WindowEvent e) {
    }
    public void windowIconified(WindowEvent e) {
    }
    public void windowDeiconified(WindowEvent e) {
    }
}
// Finestra di dialogo che presenta una griglia di pulsanti
// per cambiare il titolo del frame.
class ChangeTitleDialog extends Dialog {
    Frame fr;
    // Crea la finestra di dialogo e memorizza la stringa del titolo...
    public ChangeTitleDialog(Frame parent,boolean modal,String title) {
         // Crea una finestra di dialogo con titolo
         super(parent,title,modal);
         fr = parent;
         // Crea un layout a griglia per i pulsanti...
```

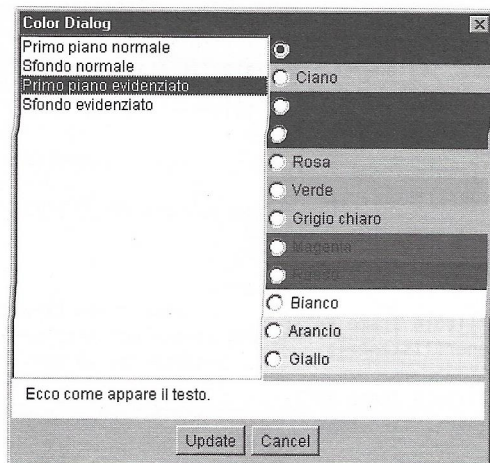
```
        setLayout(new GridLayout(3,2));
        // Aggiunge i pulsanti per scegliere i titoli
        DialogTitleListener listener = new DialogTitleListener(fr);
        Button b = new Button("Titolo frame#1");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #2");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #3");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #4");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #5");
        b.addActionListener(listener);
        add(b);
        b = new Button("Titolo frame #6");
        b.addActionListener(listener);
        add(b);
        // Un ascoltatore per chiduere la finestra di dialogo...
        addWindowListener(new DialogListener(this));
        // Visualizzazione...
        pack();
        resize(300,200);
    }
}
```

L'esempio della finestra di dialogo dei colori

Nel seguito viene creata una finestra di dialogo dei colori per spiegare ulteriormente l'utilizzo della classe `Dialog`. L'esempio permette di associare un colore al testo in primo piano e sullo sfondo. La Figura 22.1 mostra come viene visualizzata tale finestra di dialogo.

Utilizzo della finestra di dialogo dei colori

Quando la finestra di dialogo dei colori viene visualizzata, nella parte superiore sinistra vengono mostrati gli elementi del testo che possono essere modificati. Questi sono il colore in primo piano normale, il colore di sfondo normale, il colore in primo piano evidenziato e il colore di sfondo evidenziato. Quando si seleziona un elemento nell'elenco, viene evidenziato il pulsante di opzione del colore corrispondente. Il testo sotto l'elenco indica come apparirebbe il testo con i colori specificati. Il testo viene visualizzato nello stato normale (con entrambi i colori in primo piano e di sfondo) o sullo sfondo. Se si seleziona un nuovo colore con i pulsanti di opzione, l'oggetto `Canvas` viene aggiornato in modo da visualizzare la nuova combinazione di colori. Quando si seleziona il pulsante `Update`, il testo viene aggiornato con le impostazioni dei colori per l'elemento corrente dell'elenco.

Figura 22.1 *La finestra di dialogo dei colori.*

Costruzione della finestra di dialogo dei colori

Per creare la finestra di dialogo dei colori vengono utilizzate quattro classi. La classe `ChooseColorDialog` è una sottoclasse di `Dialog` e controlla la visualizzazione principale della finestra di dialogo. La classe `colorDisplay` è una classe derivata da `Canvas` che disegna il testo con i colori corrispondenti ai colori di sfondo e in primo piano selezionati. La classe `ColoredCheckbox` disegna una casella di controllo associata a un determinato oggetto `Color`; lo sfondo della casella è visualizzato con quel colore. La classe `ColoredCheckboxGroup` raggruppa gli elementi `ColoredCheckbox` in modo che funzionino come parte del gruppo di pulsanti radio. Oltre a queste classi principali, vi sono tre classi di ricevitori connesse alla finestra di dialogo.

L'esame della finestra di dialogo dei colori inizia dai suoi componenti. Il Listato 22.3 contiene il codice per la classe `ColoredCheckbox`, la cui funzionalità più interessante è che associa se stessa a un determinato oggetto `Color`. Lo sfondo della finestra di dialogo viene visualizzato in base all'oggetto `Color` e, utilizzando il metodo `setIfColorMatches()`, viene selezionata la casella di controllo se il colore inviato corrisponde al colore interno della finestra di dialogo. La casella di controllo in questo caso è un pulsante di opzione, in quanto la classe è associata a un oggetto `CheckboxGroup`. Gli oggetti caselle di controllo hanno pulsanti di opzione solo se sono associati a un oggetto `CheckboxGroup`; in un gruppo è possibile selezionare un solo pulsante di opzione alla volta. Se per un oggetto casella di controllo non viene specificato alcun gruppo, non vi sono restrizioni sul numero di caselle di controllo che possono essere selezionate.

Listato 22.3 *La classe ColoredCheckbox.*

```
// Classe per creare una casella di controllo associata
// a un dato colore...
class ColoredCheckbox extends Checkbox {
   Color color;  // Il colore della casella di controllo...
```

```
    // Il costruttore crea una casella di controllo con il colore specificato...
    public ColoredCheckbox(Color color, String label,
          CheckboxGroup grp, boolean set) {
          // Richiama il costruttore di default...
          super(label,grp,set);
          this.color = color;
          setBackground(color);
    }
    // Imposta se stesso a true se il colore corrisponde...
    public void setIfColorMatches(Color match) {
       if (color == match)
          setState(true);
       else
          setState(false);
    }
    // Restituisce il colore corrispondente alla casella...
    public Color getColor() {
       return color;
    }
 }
```

La classe `ColoredCheckboxGroup` viene utilizzata per contenere oggetti `ColoredCheckbox`. Il suo costruttore crea un numero preselezionato di caselle di controllo colorate associate a un oggetto `Panel`. Di seguito sono riportate le prime righe della dichiarazione della classe `ColoredCheckboxGroup`:

```
class ColoredCheckboxGroup extends CheckboxGroup {
   // Array per contenere caselle di controllo...
   ColoredCheckbox c[] = new ColoredCheckbox[12];
   // Costruttore. Crea le caselle di controllo senza
   // alcun colore di default scelto...
   public ColoredCheckboxGroup(Panel p,ItemListener listener) {
      // Richiama il costruttore di default...
      super();
      // Crea le caselle di controllo e le memorizza
      // in un pannello e in un array di riferimento...
      c[0] = new ColoredCheckbox(Color.black,"Nero",this,false);
      c[0].addItemListener(listener);
      p.add(c[0]);
      c[1] = new ColoredCheckbox(Color.cyan,"Ciano",this,false);
      c[1].addItemListener(listener);
      p.add(c[1]);
```

`ColoredCheckboxGroup` non è un oggetto `Container` e pertanto le caselle di controllo devono essere associate a un oggetto `Panel` in modo da soddisfare di volta in volta le diverse necessità. Si noti l'utilizzo delle costanti `Color` nella costruzione degli oggetti `ColoredCheckbox`. L'array di riferimento (variabile c) viene utilizzato nell'altro metodo della classe, `setMatchingColor()`, per impostare il pulsante di opzione dell'oggetto `ColoredCheckbox` che corrisponde a un determinato colore:

```
public void setMatchingColor(Color match) {
      for (int i = 0; i < c.length; ++i)
         c[i].setIfColorMatches(match);
   }
```

Poiché gli oggetti ColoredCheckbox vengono identificati in base al colore, con questa tecnica si evita un lungo lavoro di codifica dei nomi dei colori per determinare quale pulsante di opzione dovrebbe essere attivato.

A ogni casella di controllo è collegata una classe ItemListener. Questo ascoltatore viene richiamato ogni volta che viene selezionata una casella di controllo. Come viene spiegato nel paragrafo successivo, l'ascoltatore viene utilizzato per impostare i colori della finestra di dialogo ogni volta che viene selezionata la casella di controllo di un colore.

La classe colorDisplay deriva da Canvas e disegna del testo (specificato nella variabile stringa displayText) con i colori di sfondo e in primo piano specificati. Il Listato 22.4 pone in rilievo alcune delle funzionalità più interessanti della classe colorDisplay.

Listato 22.4 *Parti della classe colorDisplay.*

```
// Il layout lo richiama per determinare la dimensione minima
// dell'oggetto.  In questo caso, occorre che sia grande
// abbastanza per fare spazio al testo da visualizzare...
   public Dimension minimumSize() {
      // Determina la metrica del font corrente...
      FontMetrics fm = getFontMetrics(getFont());
      return new Dimension(fm.stringWidth(displayText),
         2 * fm.getHeight());
   }
   // Disegna colori e testo...
   public synchronized void paint(Graphics g) {
      if ((foreground == null) || (background == null))
         return;
      // Imposta lo sfondo...
      Dimension dm = size();
      g.setColor(background);
      g.fillRect(0,0,dm.width,dm.height);
      // Disegna la stringa
      g.setColor(foreground);
      // Imposta le dimensioni. Un po' a sinistra
      FontMetrics fm = getFontMetrics(getFont());
      int x = fm.charWidth('W');
      // E al centro verticalmente...
      int y = fm.getHeight();
      g.drawString(displayText,x,y);
   }
```

Il metodo paint() disegna il canvas se sono stati selezionati i colori di sfondo e di primo piano. Il metodo inizia con l'ottenere le dimensioni dell'area di disegno per mezzo del metodo size() (una parte standard delle sottoclassi di Component), quindi riempie lo sfondo con il colore utilizzando i metodi setColor() e fillRect() della classe Graphics e successivamente imposta il colore del testo da visualizzare (primo piano). Ottenendo il FontMetrics corrente, il canvas può calcolare una posizione corretta per la stringa di testo; il metodo getHeight() restituisce l'altezza complessiva del carattere e il metodo drawString() disegna quindi il testo nella posizione specificata.

Il metodo `minimumSize()` viene utilizzato con i layout, discussi nel Capitolo 21. Quando l'AWT crea la visualizzazione di un gruppo di componenti, collabora con i layout per decidere la posizione e le dimensioni dei componenti stessi. A volte si desidera che un componente contribuisca alla definizione delle sue dimensioni. A tale scopo è possibile richiamare due metodi della classe `Component`: il metodo `preferredSize()` restituisce le dimensioni preferite del componente, mentre il metodo `minimumSize()` restituisce le dimensioni minime con cui il componente può essere creato. Per la classe `colorDisplay()`, il metodo `minimumSize()` restituisce le dimensioni minime del componente. Per la classe `colorDisplay()`, il metodo `minimumSize()` restituisce l'informazione che il componente dovrebbe essere sufficientemente grande da visualizzare la stringa di testo e alto almeno il doppio del carattere corrente. Il metodo fa tutto ciò ottenendo il `FontMetrics` del carattere corrente e richiamando rispettivamente i metodi `stringWidth()` e `getHeight()`.

Infine, la finestra di dialogo è pronta per essere creata. Il Listato 22.5 mostra il codice completo per la finestra di dialogo dei colori. Il metodo `createComponents()` aggiunge i componenti nella finestra di dialogo utilizzando la classe `GridBagLayout`. La cosa più importante che avviene qui è l'impostazione del controllo dell'elenco in modo che utilizzi più di metà della finestra di dialogo e l'inserimento delle caselle di controllo dei colori nell'altra metà (Figura 22.1). Lo scopo principale è quello di impostare le variabili `weighty` e `gridheight` di `GridBagConstraints` sui valori appropriati. Impostando la prima su `1.0`, si indica al layout che i componenti associati devono essere prominenti per quanto concerne l'altezza del layout. Quando `weighty` è impostato su `0.0`, viene data minore priorità all'altezza dei componenti corrispondenti.

Il metodo `preferredSize()` del Listato 22.5 restituisce le dimensioni desiderate della finestra di dialogo. Questa dovrebbe essere larga 3 volte la stringa più lunga nel componente dell'elenco e alta 24 volte il carattere corrente. Con queste impostazioni, la finestra di dialogo dovrebbe contenere tutto comodamente.

Listato 22.5 *La costruzione della classe ChooseColorDialog.*

```
// Finestra di dialogo per la scelta dei colori...
class ChooseColorDialog extends Dialog {
   ColoredCheckboxGroup colorGrp;  // Per pulsanti di opzione dei colori...
   List choiceList;  // Elenco dei colori...
   colorDisplay d; // Visualizzazione del testo...

   // Definizioni per l'elenco...
   static final int NORMAL_FORE = 0;
   static final int NORMAL_BACK = 1;
   static final int HILITE_FORE = 2;
   static final int HILITE_BACK = 3;
   Color normalForeColor;
   Color normalBackColor;
   Color highlightedForeColor;
   Color highlightedBackColor;
   // Costruisce la finestra di dialogo per la scelta dei colori...
   public ChooseColorDialog(Frame parent,boolean modal) {
```

```
        // Crea la finestra di dialogo con titolo
        super(parent,"Color Dialog",modal);
        // Inizializza i colori...
        normalForeColor = Color.black;
        normalBackColor = Color.white;
        highlightedForeColor = Color.black;
        highlightedBackColor = Color.white;
        // Crea i componenti...
        createComponents();
        pack();  // Compatta...
        // Ridimensiona...
        resize(preferredSize());
    }
    // Il layout richiama questo per determinare la dimensione preferita
    // della finestra di dialogo. Deve essere grande abbastanza per il testo,
    // le caselle, il canvas e i pulsanti...
    public Dimension preferredSize() {
        // Determina la metrica del font corrente...
        FontMetrics fm = getFontMetrics(getFont());
        int width = 3 * fm.stringWidth("Highlighted foreground");
        int height = 24 * fm.getHeight();
        return new Dimension(width,height);
    }
    // Crea il pannello di visualizzazione principale...
    void createComponents() {
     // Utilizza GridBagConstraints...
     GridBagLayout g = new GridBagLayout();
     setLayout(g);
     GridBagConstraints gbc = new GridBagConstraints();
     // Imposta i vincoli per gli oggetti superiori...
     gbc.fill = GridBagConstraints.BOTH;
     gbc.weightx = 1.0;
     gbc.weighty = 1.0;
     gbc.gridheight = 10;
     // Aggiunge l'elenco di scelte...
     choiceList = new List();
     choiceList.addItem("Primo piano normale");
     choiceList.addItem("Sfondo normale");
     choiceList.addItem("Primo piano evidenziato");
     choiceList.addItem("Sfondo evidenziato");
     g.setConstraints(choiceList,gbc);
     add(choiceList);
     // Aggiunge gli ascoltatori...
     ColorListboxListener listListener = new ColorListboxListener(this);
     choiceList.addItemListener(listListener);
     // Crea il pannello delle caselle di controllo
     Panel checkboxPanel = new Panel();
     checkboxPanel.setLayout(new GridLayout(12,1));
     // Crea il gruppo e i pulsanti di opzione...
     colorGrp = new ColoredCheckboxGroup(checkboxPanel, new
                 ColorCheckboxListener(this));
     colorGrp.setMatchingColor(Color.magenta);
     // Crea e aggiunge il pannello a destra...
     gbc.gridwidth = GridBagConstraints.REMAINDER;
     g.setConstraints(checkboxPanel,gbc);
     add(checkboxPanel);
```

```
    // Visualizza il colore scelto...
    d = new colorDisplay("Ecco come appare il testo.");
    // Lo aggiunge al layout...
    gbc.weighty = 0.0;
    gbc.weightx = 1.0;
    gbc.gridwidth = GridBagConstraints.REMAINDER;
    gbc.gridheight = 1;
    g.setConstraints(d,gbc);
    add(d);
    // Dua pulsanti: "Update" e "Cancel"
    Panel p = new Panel();
    ChooseActionListener listener = new ChooseActionListener(this);
    Button b = new Button("Update");
    b.addActionListener(listener);
    p.add(b);
    b = new Button("Cancel");
    b.addActionListener(listener);
    p.add(b);
    // Aggiunge al layout...
    gbc.gridwidth = GridBagConstraints.REMAINDER;
    g.setConstraints(p,gbc);
    add(p);
  }
  // Imposta i default alla visualizzazione...
  public synchronized void show() {
     super.show(); // Call the default show method...
     // Imposta il default per l'elenco...
     choiceList.select(0);
  }
  // Aggiorna i colori globali con le impostazioni correnti...
  public void update()
  {
     // Imposta i colori del canvas in base allo stato...
     int index = choiceList.getSelectedIndex();
     switch (index) {
       case NORMAL_FORE:
           normalForeColor = d.getForeColor();
           break;
       case NORMAL_BACK:
           normalBackColor = d.getBackColor();
           break;
       case HILITE_FORE:
           highlightedForeColor = d.getForeColor();
           break;
       case HILITE_BACK:
           highlightedBackColor = d.getBackColor();
           break;
       default:
           break;
     }
  }
  // Imposta i colori del titolo e della scelta
  // in base al valore corrente nell'elenco...
  public void selectedChoiceListItem() {
     Color fore,back;  // Visualizza i colori del canvas
     // Imposta i colori del canvas in base allo stato...
     int index = choiceList.getSelectedIndex();
```

```
        if ((index == NORMAL_FORE) || (index == NORMAL_BACK)) {
            fore = normalForeColor;
            back = normalBackColor;
        }
        // Altrimenti è lo sfondo...
        else {
            fore = highlightedForeColor;
            back = highlightedBackColor;
        }
        // Aggiorna il canvas...
        d.setForeColor(fore);
        d.setBackColor(back);
        d.repaint();
        // Aggiorna i pulsanti di opzione dei colori...
        Color radioColor;
        // Numeri pari in primo piano, dispari sfondo...
        if ((index % 2) == 0)
            radioColor = fore;
        else
            radioColor = back;
        colorGrp.setMatchingColor(radioColor);
    }

    // Aggiorna il canvas quando il pulsante del colore cambia...
    public void selectedRadioItem() {
        // Ottiene la scelta dalla casella dei colori...
        ColoredCheckbox box = (ColoredCheckbox)colorGrp.getCurrent();
        Color color = box.getColor();
        // Se è normale imposta i colori del canvas...
        Color fore,back;
        int index = choiceList.getSelectedIndex();
        // Imposta il colore di sfondo o di primo piano in base
        // alla selezione corrente nell'elenco...
        if ((index == NORMAL_FORE) || (index == HILITE_FORE))
            d.setForeColor(color);
        else
            d.setBackColor(color);
        // Ridisegna il canvas...
        d.repaint();
    }
}
```

Utilizzo della finestra di dialogo

Per controllare le azioni della finestra di dialogo vengono utilizzate tre classi di ascoltatori, che sono connessi alla finestra di dialogo nel metodo `createComponents()` della classe `ChooseColorDialog`. La prima classe, `ColorListboxListener`, viene richiamata ogni volta che viene selezionato un elemento dell'elenco. Questa classe si basa sull'interfaccia `ItemListener`, che richiama il metodo `itemStateChanged()` quando nell'elenco viene selezionato un elemento. La classe `ColorCheckboxListener` è simile all'interfaccia `ItemListener`, a eccezione del fatto che viene richiamata quando vengono selezionate le caselle di controllo (in questo caso i pulsanti di opzione).

Questa classe informa la finestra di dialogo che è stato visualizzato un nuovo colore. Infine, la classe `ChooseActionListener` viene utilizzata per indicare quale dei due pulsanti è stato selezionato: il pulsante `Update` imposta permanentemente un colore, mentre il pulsante `Cancel` fa sì che la finestra di dialogo venga distrutta.

Listato 22.6 *Le classi di ascoltatori della finestra di dialogo dei colori.*

```
// Ascoltatori per la finestra di dialogo dei colori
// Questo è per l'elenco...
class ColorListboxListener implements ItemListener {
   ChooseColorDialog dlg; // La finestra di dialogo dei colori...
   // Costruttore...
   public ColorListboxListener(ChooseColorDialog dlg) {
        this.dlg = dlg;
   }
   // Gestisce l'azione da eseguire...
   public void itemStateChanged(ItemEvent e) {
       dlg.selectedChoiceListItem();
   }
}
// Questo è per le caselle di controllo...
class ColorCheckboxListener implements ItemListener {
   ChooseColorDialog dlg; // La finestra di dialogo dei colori...
   // Costruttore...
   public ColorCheckboxListener(ChooseColorDialog dlg) {
        this.dlg = dlg;
   }
   // Gestisce l'azione da eseguire...
   public void itemStateChanged(ItemEvent e) {
       dlg.selectedRadioItem();
   }
}
class ChooseActionListener implements ActionListener {
   // Azioni sulla finestra di dialogo...
   ChooseColorDialog dlg; // La finestra dei colori...
   // Costruttore...
   public ChooseActionListener(ChooseColorDialog dlg) {
      this.dlg = dlg;
   }
   // Gestisce l'azione da eseguire...
   public void actionPerformed(ActionEvent e) {
       // Determina l'azione eseguita...
       String selection = e.getActionCommand();
       if (selection.equals("Update"))
            dlg.update();
       // Altrimenti distrugge la finestra di dialogo...
       if (selection.equals("Cancel"))
            dlg.dispose();
   }
}
```

Chiamata della finestra di dialogo

L'oggetto `Frame` è responsabile della visualizzazione della finestra di dialogo e può dichiararne una variabile nel seguente modo:

```
ChooseColorDialog colorDialog;  // Finestra del colore...
```

Nel suo costruttore, il frame crea un'istanza della finestra di dialogo dei colori con la seguente chiamata:

```
colorDialog = new ChooseColorDialog(this, true);
```

Questo segmento di codice indica che il frame è il genitore della finestra di dialogo e che questa è modale. Come già menzionato, una finestra di dialogo modale non permette l'input in altre finestre finché è visualizzata.

Le finestre di dialogo non vengono visualizzate automaticamente quando vengono create, ma per mezzo di una chiamata specifica al metodo `show()`.

L'esempio della finestra di dialogo dei caratteri

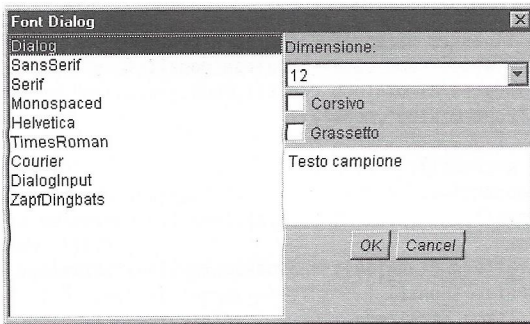
La discussione sull'esempio della finestra di dialogo dei caratteri non è così lunga come la precedente discussione sulla finestra di dialogo dei colori. Per diversi aspetti, i due esempi sono simili e pertanto non è necessaria una spiegazione dettagliata della finestra di dialogo dei caratteri.

Nella Figura 22.2 è mostrata la finestra di dialogo dei caratteri. Questa si basa sulla classe `ChooseFontDialog`, che visualizza i componenti in due colonne, in modo simile alla finestra di dialogo dei colori. La famiglia di caratteri, lo stile e le dimensioni correnti sono visualizzati in un oggetto di visualizzazione `Canvas` della classe `fontDisplay`, molto simile alla classe `colorDisplay`.

Il componente dell'elenco nella parte sinistra della finestra di dialogo mostra i caratteri disponibili nella piattaforma corrente; per ottenere queste informazioni utilizza la classe `Toolkit` dell'AWT. Di seguito è riportato il codice che crea il controllo e che lo aggiunge alle famiglie di caratteri:

```
// Aggiunge all'elenco di scelte...
   // Ottiene la selezione dal toolkit...
    choiceList = new List();
    String fontList[] = Toolkit.getDefaultToolkit().getFontList();
    for (int i = 0; i < fontList.length; ++i)
       choiceList.addItem(fontList[i]);
```

Nella finestra di dialogo viene aggiunto un menu di scelta per elencare le dimensioni dei caratteri che possono essere utilizzate. Per impostare gli stili grassetto e corsivo vengono utilizzate due caselle di controllo. Se nessuna di queste opzioni è selezionata, lo stile del carattere è quello di base.

Figura 22.2 *La finestra di dialogo dei caratteri.*

Ogni volta che uno di questi controlli cambia, il carattere visualizzato viene aggiornato in modo da riflettere il cambiamento, che avviene nel metodo `paintSample()`, riportato nel Listato 22.7.

Listato 22.7 *Il codice sorgente completo per la finestra di dialogo dei caratteri.*

```
// Ascoltatori per la finestra di dialogo dei caratteri
// Questo è per tutte le routine di ridisegno...
class FontItemListener implements ItemListener {
   ChooseFontDialog dlg; // La finestra dei colori...
   // Costruttore...
   public FontItemListener(ChooseFontDialog dlg) {
        this.dlg = dlg;
   }
   // Gestisce l'azione da eseguire...
   public void itemStateChanged(ItemEvent e) {
       dlg.paintSample();
   }
}
class ChooseActionListener implements ActionListener {
   // Azioni sulla finestra...
   ChooseFontDialog dlg; // La finestra dei colori...
   // Costruttore...
   public ChooseActionListener(ChooseFontDialog dlg) {
      this.dlg = dlg;
   }
   // Chiude tutto...
   public void actionPerformed(ActionEvent e) {
      dlg.dispose();
   }
}
// Finestra di dialogo per la scelta dei colori...
class ChooseFontDialog extends Dialog {
   List choiceList;  // Elenco dei colori...
   fontDisplay d; // Visualizzazione del testo...
   Choice choiceSize;  // Dimensione del carattere...
   Checkbox checkItalics;
   Checkbox checkBold;
   Frame fr;
   Font currentFont;  // Carattere corrente in esempio...
```

```
Font defaultFont; // Carattere di default...
// Costruisce la finestra di dialogo per la scelta dei colori...
public ChooseFontDialog(Frame parent,boolean modal) {
   // Crea una finestra di dialogo con titolo
   super(parent,"Font Dialog",modal);
   fr = parent;
   defaultFont = getFont();
   // Crea i componenti...
   createComponents();
}
// Ottiene il carattere di default e imposta la visualizzazione...
private void setDefaultFont() {
   // Ottiene l'ultimo carattere...
   currentFont = getFont();
   // Ottiene l'elenco di caratteri...
   String s = currentFont.getName();
   int index = findListString(choiceList,s);
   if (index >= 0)
         choiceList.select(index);
   else
         choiceList.select(0);
   // Ottiene la dimensione di default
   int sizeFont = currentFont.getSize();
   index = findChoiceString(choiceSize,
      String.valueOf(sizeFont));
   if (index >= 0)
         choiceSize.select(index);
   else
         choiceSize.select(0);
   // Imposta lo stile...
   int styleFont = currentFont.getStyle();
   if ((styleFont & Font.BOLD) != 0)
          checkBold.setState(true);
   else
          checkBold.setState(false);
   if ((styleFont & Font.ITALIC) != 0)
          checkItalics.setState(true);
   else
          checkItalics.setState(false);
   // Imposta lo stile del canvas...
   d.setFont(currentFont);
}
// Il layout lo chiama per determinare la dimensione preferita
// della finestra di dialogo. Deve essere grande abbastanza per il testo,
// le caselle di controllo, il canvas e i pulsanti...
public Dimension preferredSize() {
   // Ottiene la metrica del carattere corrente...
   FontMetrics fm = getFontMetrics(getFont());
   int width = 3 * fm.stringWidth("Highlighted foreground");
   int height = 14 * fm.getHeight();
   return new Dimension(width + 40,height + 40);
}
// Crea il pannello di visualizzazione principale...
private void createComponents() {
 // Utilizza GridBagConstraints...
 GridBagLayout g = new GridBagLayout();
```

```
setLayout(g);
GridBagConstraints gbc = new GridBagConstraints();
// Imposta i vincoli per gli oggetti superiori...
gbc.fill = GridBagConstraints.BOTH;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
gbc.gridheight = 10;
// Aggiunge l'elenco di scelte...
// Determina la selezione dal toolkit...
choiceList = new List();
String fontList[] = Toolkit.getDefaultToolkit().getFontList();
for (int i = 0; i < fontList.length; ++i) {
   choiceList.addItem(fontList[i]);
} // end if
g.setConstraints(choiceList,gbc);
add(choiceList);
// Imposta i valori di default...
gbc.weighty = 0.0;
gbc.weightx = 1.0;
gbc.gridheight = 1;
gbc.gridwidth = GridBagConstraints.REMAINDER;
// Crea un'etichetta da visualizzare...
Label l = new Label("Dimensione:");
// Add to grid bag...
g.setConstraints(l,gbc);
add(l);
// Crea la casella di scelta...
choiceSize = new Choice();
choiceSize.addItem("8");
choiceSize.addItem("10");
choiceSize.addItem("12");
choiceSize.addItem("14");
choiceSize.addItem("16");
choiceSize.addItem("20");
// La aggiunge al layout...
g.setConstraints(choiceSize,gbc);
add(choiceSize);
// Aggiunge Corsivo...
checkItalics = new Checkbox("Corsivo");
g.setConstraints(checkItalics,gbc);
add(checkItalics);
// Aggiunge Grassetto...
checkBold = new Checkbox("Grassetto");
g.setConstraints(checkBold,gbc);
add(checkBold);
// Imposta gli ascoltatori...
FontItemListener itemListener = new FontItemListener(this);
choiceList.addItemListener(itemListener);
choiceSize.addItemListener(itemListener);
checkItalics.addItemListener(itemListener);
checkBold.addItemListener(itemListener);
// Visualizza il colore scelto...
d = new fontDisplay("Testo campione");
// Add to grid bag...
g.setConstraints(d,gbc);
add(d);
```

```
    // Due pulsanti: "OK" e "Cancel"
    Panel p = new Panel();
    ChooseActionListener actionListener = new ChooseActionListener(this);
    Button b = new Button("OK");
    b.addActionListener(actionListener);
    p.add(b);
    b = new Button("Cancel");
    b.addActionListener(actionListener);
    p.add(b);
    // Aggiunge al layout...
    gbc.gridwidth = GridBagConstraints.REMAINDER;
    g.setConstraints(p,gbc);
    add(p);
   }
    // Imposta i default per la visualizzazione...
    public void show() {
      setFont(defaultFont);
      // Imposta i default...
      setDefaultFont();
      pack(); // Compatta...
      // Ridimensiona...
      resize(preferredSize());
      // Imposta la finestra di dialogo del carattere...
      super.show(); // Richiama il costruttore di default...
}
   // Imposta il canvas in modo che mostri
   // il carattere attualmente selezionato
   public synchronized void paintSample() {
      // Determina la famiglia
      String fontName = choiceList.getSelectedItem();
      // Determina la dimensione
      String fontSize = choiceSize.getSelectedItem();
      // Imposta lo stile
      int fontStyle = Font.PLAIN;
      if (checkItalics.getState())
            fontStyle += Font.ITALIC;
      if (checkBold.getState())
            fontStyle += Font.BOLD;
      // Crea un font con gli attributi appropriati...
      currentFont = new Font(fontName,fontStyle,
            Integer.parseInt(fontSize));
      // Imposta il nuovo font sul canvas...
      d.setFont(currentFont);
      // Ridisegna per mostrare il nuovo carattere..
      d.repaint();
   }
   // Restituisce l'indice della stringa nell'elenco...
   // -1 significa non trovato
   public int findListString(List l,String s) {
      for (int i = 0; i < l.countItems(); ++i) {
         if (s.equals(l.getItem(i)) )
             return i;
      }
      return -1;
   }
   // Restituisce l'indice della stringa nella scelta...
```

```
    // -1 means not found
    public int findChoiceString(Choice c,String s) {
       for (int i = 0; i < c.countItems(); ++i) {
          if (s.equals(c.getItem(i)) )
             return i;
       }
       return -1;
    }
}
// Una piccola classe che illustra
// l'impostazione corrente per evidenziazione e sfondo
class fontDisplay extends Canvas {
    String displayText;
    // Costruisce la visualizzazione memorizzando
    // il testo da mostrare...
    public fontDisplay(String displayText) {
       super();
       this.displayText = displayText;
    }
    // Il layout lo richiama per ottenere la dimensione minima
    // dell'oggetto. In questo caso deve essere grande abbastanza
    // per fare spazio al testo da visualizzare...
    public Dimension minimumSize() {
       // Ottiene la metrica del carattere corrente...
       FontMetrics fm = getFontMetrics(getFont());
       return new Dimension(fm.stringWidth(displayText),
          4 * fm.getHeight());
    }
    // Disegna colori e testo...
    public synchronized void paint(Graphics g) {
       // Imposta lo sfondo...
       Dimension dm = size();
       g.setColor(Color.white);
       g.fillRect(0,0,dm.width,dm.height);
       // Disegna la stringa
       g.setColor(Color.black);
       // Imposta le dimensioni. Appena a sinistra...
       FontMetrics fm = getFontMetrics(getFont());
       int x = fm.charWidth('I');
       // E centrato in altezza...
       int y = fm.getHeight();
       g.drawString(displayText,x,y);
    }
}
```

La classe FileDialog

La classe `FileDialog` è una sottoclasse di `Dialog` e viene utilizzata per fornire un approccio indipendente dalla piattaforma che permetta agli utenti di selezionare i file da caricare o da salvare. Le istanze della classe `FileDialog` di norma rispecchiano le convenzioni della GUI sottostante. Ad esempio, un oggetto `FontDialog` per caricare un file nell'ambiente Windows 95 segue le convenzioni della finestra di dialogo `Apri` di Windows 95.

L'oggetto `FileDialog` può essere creato nella modalità di caricamento o di salvataggio dei file. Queste due finestre di dialogo sono molto simili, ma eseguono operazioni leggermente diverse. Ad esempio, se viene specificato un nome di un file preesistente, la versione `Salva` della finestra di dialogo notifica all'utente che il file esiste. Di seguito viene presentato il codice necessario per creare finestre di dialogo per caricare o salvare file:

```
FileDialog openFileDialog;  // Apertura di file...
FileDialog saveFileDialog;  // Salvataggio di file...
openFileDialog = new FileDialog(this,"Apri", FileDialog.LOAD);
saveFileDialog = new FileDialog(this,"Salva", FileDialog.SAVE);
```

Il flag intero nell'ultimo parametro del costruttore indica la modalità di funzionamento della finestra di dialogo. Quando viene visualizzata per mezzo del metodo `show()`, la finestra di dialogo agisce in modo modale, cosicché l'input nel frame è bloccato finché la finestra di dialogo rimane visualizzata. Dopo che è stata effettuata una scelta, il file e la directory selezionati possono essere richiamati per mezzo dei metodi `getFile()` e `getDirectory()`. Il metodo `getFile()` restituisce un valore `null` se l'utente annulla la finestra di dialogo.

È interessante notare che gli oggetti `FileDialog` creati in questo capitolo non vengono visualizzati quando l'applet viene eseguito in Netscape Navigator. Per motivi di sicurezza, Netscape non permette ai metodi che si basano su file di essere chiamati da un applet. Nell'ultima versione di Microsoft Internet Explorer si ottiene un'eccezione di sicurezza perfino se si tenta di fare riferimento alla classe `FileDialog`. Di conseguenza, questa classe è utile solamente per le applicazioni autonome.

Riepilogo

Come hanno mostrato gli esempi di questo capitolo, la maggior parte del lavoro per creare una finestra di dialogo consiste nell'impostare i controlli dei componenti. Le tecniche sono molto simili a quelle utilizzate per creare controlli in un applet, con la differenza che potrebbe rendersi necessario richiamare il metodo `pack()` o il metodo `preferredSize()` per conferire alla finestra di dialogo l'aspetto desiderato.

L'altro problema importante con le finestre di dialogo è la sicurezza. Alcuni browser, come Netscape Navigator, considerano le finestre di dialogo una possibile minaccia e, per questo motivo, nella barra di stato del browser può essere visualizzato un avvertimento o un messaggio che indica che la finestra visualizzata non è fidata. Nel peggiore dei casi, la finestra di dialogo potrebbe non essere addirittura visualizzata o potrebbe riportare un flag con un'eccezione di sicurezza (come nel caso della classe `FileDialog` in Microsoft Internet Explorer). Per questi motivi, è necessario fare attenzione quando si inseriscono finestre di dialogo negli applet. In base a quanto detto, le finestre di dialogo risultano più adatte per le applicazioni autonome di Java.